

Revision History for //depot/main/redirector/MAPIMailbox.cpp

Revision	File Name	Changelist	Date	Description
3	//depot/m...	2357	2000/01/14	Merge in calendar sync changes from Continuus.
2	//depot/m...	2265	2000/01/13	Apply 2.0 files
1	//depot/m...	2100	2000/01/10	Add redirector to P4

Actions

Display Branching History

```
*****  
MAPIMailbox.cpp  
*****  
  
5 (C) 1997 Research In Motion Ltd.  
*****  
  
10 // Recurrence blob related structures and constants  
*****  
  
15 #pragma warning(disable:4201)  
#pragma warning(disable:4514)  
  
20 #include "MAPIMailbox.h"  
#include "MAPIMailboxNotify.h"  
#include "MailboxManager.h"  
#include "globals.h"  
#include "rimmessage.h"  
#include "rimcalendar.h"  
#include "rimstream.h"  
#include "debug.h"  
#include "Mutex.h"  
#include "timeconversion.h"  
#include "ICSAgent.h"  
#include "CallICSAgent.h"  
#include "icalformat.h"  
#include "calsyncstate.h"  
  
25 Mutex g_DebugMutex;  
  
30 // Recurrence blob related structures and constants  
*****  
  
35 #define _SECOND ((int64)1000000)  
#define _MINUTE (60 * _SECOND)  
#define _HOUR (60 * _MINUTE)  
#define _DAY (24 * _HOUR)  
  
40 const USHORT TYPE_DAILY = 0x200A;  
const USHORT TYPE_WEEKLY = 0x200B;  
const USHORT TYPE_MONTHLY = 0x200C;  
const USHORT TYPE_YEARLY = 0x200D;
```

```

-cleanup:
try
{
    // resource cleanups
    if( !pIStreamBody ) {
        pIStreamBody->Release();
    }
}

11830    return( rc );
}

11835    }
    catch( ... )
    {
        ReportException(mailbox_fcn);
        return( rc );
    }
}

11840    } // Exception-handling
}

11845    }

// *** RIMEventtoCDOAppointment ***
// Purpose: Copies the Event properties into the corresponding members of the provided
//          CDO appointment object.
// Parameters:
// Returns:
//         

11850    static const char mailbox_fcn[] = "RIMEventtoCDOAppointment";

11855    bool MAPIMailbox::RIMEventtoCDOAppointment( RIMEEvent *pEvent, AppointmentItemPtr &spAppointmentItem )
{
    assert( pEvent );
    assert( spAppointmentItem );
    spAppointmentRecipients;
    spAppointmentAttendee;
    RecurrencePatternPtr spAppointmentRecurrence;
    RIMEEvent::FrequencyType ApptFrequency;
    bool instanceRecurrence = false;
    time_t fStartTime;
    FILETIME fStartTime;
    SYSTEMTIME stStartTime;
}

```

```

11870     DOUBLE      dStartTime;
     time_t      tEndTime;
     FILETIME    flEndTime;
     SYSTEMTIME  stEndTime;
     DOUBLE      dEndTime;
     long        lAppointmentRefId = 0;
     long        lParentRefId = 0;
     long        lDeviceSeqNum = 0;
     bool        rc = true;

11880     try
     {
        /* Get the UID (our refId) */
        RIMEvent::UIDProperty* pAppointmentUID;
        if( !pEvent->GetProperty( &pAppointmentUID ) )
        {
            DebugLog::Print( DebugLog::LOG_WARNING,
                            "MAPIMailbox::RIMMeetingToMAPIMeeting - Getting UIDProperty from RIMCalendar failed for %s.",
                            m_Name );
            rc = false;
            goto _cleanup;
        }
        // if
        AppointmentRefId = pAppointmentUID->GetUID();
    }

11885     /* Get the RelatedTo Id */
     RIMEvent::RelatedToIDProperty* pAppointmentRelatedTo;
     if( !pEvent->GetProperty( &pAppointmentRelatedTo ) )
     {
        lParentRefId = pAppointmentRelatedTo->GetRelatedToID();
    }

11890     /* Get the Sequence # */
     RIMEvent::SequenceProperty* pAppointmentSequenceNum;
     if( !pEvent->GetProperty( &pAppointmentSequenceNum ) )
     {
        DebugLog::Print( DebugLog::LOG_WARNING,
                            "MAPIMailbox::RIMMeetingToMAPIMeeting - Getting SequenceProperty from RIMCalendar failed for %s.",
                            m_Name );
        rc = false;
        goto _cleanup;
    }
    // if
    lDeviceSeqNum = pAppointmentSequenceNum->GetSequenceNumber();
}

11895
11900
11905
11910

```

```

/* Get the Summary */
RIMEEvent::SummaryProperty* pAppointmentSummary;
if( pEvent->GetProperty( &pAppointmentSummary ) )
{
    spAppointmentItem->PutSubject( _variant_t(pAppointmentSummary->GetSummary()) );
}

/* Get the Location */
RIMEEvent::LocationProperty* pAppointmentLocation;
if( pEvent->GetProperty( &pAppointmentLocation ) )
{
    spAppointmentItem->PutLocation( _variant_t(pAppointmentLocation->GetLocation()) );
}

/* Get the Description */
RIMEEvent::DescriptionProperty* pAppointmentDescription;
if( pEvent->GetProperty( &pAppointmentDescription ) )
{
    RIMStream DescriptionStream = pAppointmentDescription->GetDescription();
    DescriptionStream.Seek(0); // Rewind
    unsigned int BodyLength = pAppointmentDescription->GetAvailableSize();
    unsigned char *pBodyText = new unsigned char[ BodyLength+1 ];
    memset(pBodyText, 0, BodyLength);
    DescriptionStream.Read(pBodyText, BodyLength);
    pBodyText[BodyLength] = '\0';
    _bstr_t bstrBody((const char *)pBodyText);
    spAppointmentItem->PutText(bstrBody);
    delete []pBodyText;
}

else
{
    // Set a blank message body
    spAppointmentItem->PutText(_bstr_t(""));
} //if

/* Get the Trigger */
RIMEEvent::TriggerProperty* pAppointmentTrigger;
if( pEvent->GetProperty( &pAppointmentTrigger ) )
{
    spAppointmentItem->PutReminderSet(_variant_t(1L,VT_I4));
    spAppointmentItem->PutReminderMinutesBeforeStart( _variant_t(pAppointmentTrigger->GetTrigger()) );
}

/* Get the Date/TimeStart */
RIMEEvent::DateTimeStartProperty* pAppointmentDateTimeStart;
if( !pEvent->GetProperty( &pAppointmentDateTimeStart ) )

```

```

11955     {
11956         DebugLog::Print( DebugLog::LOG_WARNING,
11957                         "MAPIMailbox::RIMMeetingToMAPIMeeting - Getting DateTimeStartProperty from RIMCalendar failed for %s.",
11958                         m_Name );
11959         rc = false;
11960         goto _cleanup;
11961     } //if
11962     // The long road to convert this to a variant time
11963     iStartTime = pAppointmentDateTimeStart->GetTime();
11964     UnixTimeToFileTime( iStartTime, &iStartTime );
11965     FileTimeToSystemTime( &iStartTime, &stStartTime );
11966     SystemTimeToVariantTime( &stStartTime, &dtStartTime );
11967     spAppointmentItem->PutStartTime( _variant_t( dtStartTime, VT_DATE ) );
11968     /* Get the Date TimeEnd */
11969     RIMEEvent::DateTimeEndProperty* pAppointmentDateTimeEnd;
11970     if( !pEvent->GetProperty( &pAppointmentDateTimeEnd ) )
11971     {
11972         DebugLog::Print( DebugLog::LOG_WARNING,
11973                         "MAPIMailbox::RIMMeetingToMAPIMeeting - Getting DateTimeEndProperty from RIMCalendar failed for %s.",
11974                         m_Name );
11975         rc = false;
11976         goto _cleanup;
11977     } //if
11978     // The long road to convert this to a variant time
11979     iEndTime = pAppointmentDateTimeEnd->GetTime();
11980     UnixTimeToFileTime( iEndTime, &iEndTime );
11981     FileTimeToSystemTime( &iEndTime, &stEndTime );
11982     SystemTimeToVariantTime( &stEndTime, &dEndTime );
11983     spAppointmentItem->PutEndTime( _variant_t( dEndTime, VT_DATE ) );
11984     /* Get the Attendee(s) */
11985     RIMEEvent::AttendeeProperty* pAppointmentAttendee;
11986     spAppointmentRecipients = spAppointmentItem->GetRecipients();
11987     pEvent->GetProperty( &pAppointmentAttendee );
11988     while( pAppointmentAttendee )
11989     {
11990         bstr_t bstrAddress = "SMTP:";
11991         bstrAddress += bstr_t( pAppointmentAttendee->GetAddress() );
11992         spAppointmentRecipients->Add( _variant_t( pAppointmentAttendee->GetDisplayName() ), _variant_t( bstrAddress ), _variant_t( (long)CdoTo ) );
11993     } //while
11994     /* Get the Recurrence Pattern, if any */
11995     RIMEEvent::RecurrenceRuleProperty* pAppointmentRecurrence;
11996     if( !pEvent->GetProperty( &pAppointmentRecurrence ) )

```

```

12000
{
    spAppointmentRecurrence = spAppointmentItem->GetRecurrencePattern();
    // Following items set no matter what the recurrence type is
    spAppointmentRecurrence->PutOccurrences(_variant_t(pAppointmentRecurrence->GetCount()));
    spAppointmentRecurrence->PutInterval(_variant_t(pAppointmentRecurrence->GetInterval()));
    ApptFrequency = pAppointmentRecurrence->GetFrequency();
    // Items that may or may not be present based on recurrence type
    if( pAppointmentRecurrence->GetDay() != RIMEvent::UNDEFINED_DAY )
    {
        spAppointmentRecurrence->PutDayOfWeekMask(_variant_t(pAppointmentRecurrence->GetDay()));

        if( pAppointmentRecurrence->GetMonth() != RIMEvent::UNDEFINED_DAY )
        {
            spAppointmentRecurrence->PutMonthOfYear(_variant_t(pAppointmentRecurrence->GetMonth()));

            if( pAppointmentRecurrence->GetMonthDay() != 0 )
            {
                spAppointmentRecurrence->PutDayOfMonth(_variant_t(pAppointmentRecurrence->GetMonthDay()));

                if( pAppointmentRecurrence->GetSetPosition() != 0 )
                {
                    fInstanceRecurrence = true;
                    spAppointmentRecurrence->PutInstance(_variant_t(pAppointmentRecurrence->GetSetPosition()));
                }
            }
        }
    }
    switch(ApptFrequency)
    {
        case RIMEvent::DAILY:
            spAppointmentRecurrence->PutRecurrenceType(_variant_t((long)CdoRecurTypeDaily));
            break;
        case RIMEvent::WEEKLY:
            spAppointmentRecurrence->PutRecurrenceType(_variant_t((long)CdoRecurTypeWeekly));
            break;
        case RIMEvent::MONTHLY:
            if( fInstanceRecurrence )
                spAppointmentRecurrence->PutRecurrenceType(_variant_t((long)CdoRecurTypeMonthlyNth));
            else
                spAppointmentRecurrence->PutRecurrenceType(_variant_t((long)CdoRecurTypeMonthly));
            break;
        case RIMEvent::YEARLY:
            if( fInstanceRecurrence )
                spAppointmentRecurrence->PutRecurrenceType(_variant_t((long)CdoRecurTypeYearlyNth));
            else
                spAppointmentRecurrence->PutRecurrenceType(_variant_t((long)CdoRecurTypeYearly));
    }
}

12005
12010
12015
12020
12025
12030
12035
12040

```

```

break;
} // switch
}
catch ( _com_error &ee)
{
    bstr_t bstrSource(e.Source());
    bstr_t bstrDescription(e.Description());
    DebugLog::Printf(DebugLog::LOG_INFORMATIONAL,
        "*** CDO *** MAPIMailbox::MAPIMailbox() - Code = %08lx, Code meaning = %s, Source = %s, Description = %s.",

12045     e.Error(), e.ErrorMessage(), (LPCTSTR) bstrSource, (LPCTSTR) bstrDescription);
    rc = false;
}

12055     catch( ... )
{
    rc = false;
}

12060     cleanup:
    return(rc);
}

12065     // *** Send ***
// Purpose:
// Parameters:
//     bool MAPIMailbox::Send( RIMMessage* pMessage, LPMESSAGE* ppMAPIMessage, bool bDeleteAfterSubmit,
//     int origRefId, bool insert_text, bool insert_attach, FOLDERSEfFolder )
{
    static const char mailbox_fcn[] = "Send(ppMAPIMessage)";

12070     assert( pMessage );
    assert( ppMAPIMessage );
}

12075     LPMESSAGE    pOrigMessage = 0;
     HRESULT      hResult;
     bool         bReCode = false;
try
{
    //DebugLog::Printf(DebugLog::LOG_DEBUG, "*** MAPI *** Sending message from %s", m_Name );
    if ( origRefId != 0 )
}

```

```

14150     {
14151         static const char mailbox_fcn[] = "TransferAppointmentReceivedNotification";
14152         EntryID *pEntry = new EntryID( m_pSession, EntryIDValue , EntryIDSize );
14153         assert( pEntry );
14154
14155         // Notify user control.
14156         m_pControl->AppointmentModifiedNotification( pEntry );
14157
14158         return;
14159     }
14160
14161     void MAPIMailbox::TransferAppointmentDeletedNotification( unsigned long SourceKeyIDSize, unsigned char * SourceKeyIDValue )
14162     {
14163         static const char mailbox_fcn[] = "TransferAppointmentReceivedNotification";
14164
14165         EntryID *pSourceKey = new EntryID( 0, SourceKeyIDValue , SourceKeyIDSize );
14166         assert( pSourceKey );
14167
14168         // Notify user control.
14169         m_pControl->AppointmentDeletedNotification( pSourceKey );
14170
14171         return;
14172     }
14173
14174     // *** Synchronize ***
14175
14176     // Purpose:
14177     // Parameters: pAppointment = RIM calendar
14178     // Returns: true if appointment sync'd successfully. Otherwise, false.
14179
14180     bool MAPIMailbox::Synchronize( RIMCalendar* pAppointment )
14181
14182     static const char mailbox_fcn[] = "Synchronize";
14183
14184     #ifndef PERSONAL_BUILD // No calendar support for desktop redirector
14185
14186         ResetLastError();
14187
14188         LPMAPITABLE pContentsTable = 0;
14189         LPSRowSet pRows = 0;
14190         LPMESSAGE pRootAppointment = 0;
14191         LPMAPITABLE pAttachmentTable = 0;
14192         LPATTACH pAttachment = 0;

```

```

LPSRowSet          pAttachRows = 0;
LPSBinary          pbInStateSourceKey = 0;

14195   FolderPtr          spCalendarFolder;
          AppointmentItemPtr spAppointmentItem;
          spCalendarMessages;
          spAppointmentFields;
          FieldPtr           spAppointmentSourceKey;

14200   variant_t          vtAppointmentSourceKey;
          sSourceKeyRes;
          spvSourceKey;
          HRESULT             hResult;
          bool                bResult = false;

14205   const ULONG            nAppointmentProps = 7; // # of appointment properties
          SizedSPropTagArray(nAppointmentProps, sptaAppointmentProps) =
          {
              PR_ENTRYID,
              PR_SOURCE_KEY,
              PROP_TAG(PROP_TYPE_LONG, 0x0E23),
              PR_RIM_CAL_APPT_DURATION,
              PR_RIM_CAL_START_DATETIME,
              PR_RIM_CAL_END_DATETIME,
              PR_LAST_MODIFICATION_TIME
          };

14210   const ULONG            nAttachProps = 5; // # of attach properties
          SizedSPropTagArray(nAttachProps, sptaAttachProps) =
          {
              PR_ATTACH_LONG_FILENAME,
              PR_ATTACH_FILENAME,
              PR_ATTACH_MIME_TAG,
              PR_ATTACH_NUM,
              PR_ATTACH_METHOD
          };

14220   const ULONG            nAttachProps = 5; // # of attach properties
          SizedSPropTagArray(nAttachProps, sptaAttachProps) =
          {
              PR_ATTACH_LONG_FILENAME,
              PR_ATTACH_FILENAME,
              PR_ATTACH_MIME_TAG,
              PR_ATTACH_NUM,
              PR_ATTACH_METHOD
          };

```

```

14235     try
14236     {
14237         RIMCalendar::MethodProperty* pAppointmentMethod = 0;
14238         RIMCalendar::CalendarEventProperty* pCalendarEvent = 0;
14239         CalendarEvent;
14240         long
14241         long
14242         long
14243         long
14244         FILETIME
14245         FILETIME
14246         FILETIME
14247         FILETIME
14248         SYSTEMTIME
14249         variant_t
14250         long
14251         long
14252         bool
14253         bool
14254         bool
14255         bool
14256
14257         // Determine the request method: PUBLISH or CANCEL
14258         // Calendar method
14259         if( .pAppointment->GetProperty( &pAppointmentMethod ) )
14260         {
14261             DebugLog::Printf( DebugLog::LOG_WARNING,
14262                             "MAPIMailbox::Synchronize - Getting MethodProperty from RIMCalendar failed for %s.",
14263                             m_Name );
14264             bResult = false;
14265             goto __end;
14266         }
14267
14268         // Loop over VEVENT components contained in the calendar object
14269         if( .pAppointment->GetProperty( &pCalendarEvent ) )
14270         {
14271             DebugLog::Printf( DebugLog::LOG_WARNING,
14272                             "MAPIMailbox::Synchronize - Getting CalendarEventProperty from RIMCalendar failed for %s.",
14273                             m_Name );
14274             bResult = false;
14275             goto __end;
14276         }

```

```

14280     while( pCalendarEvent )
14281     {
14282         CalendarEvent = pCalendarEvent->GetCalendarEvent();
14283
14284         // Get the CDO calendar folder object.
14285         if( spCalendarFolder == 0 )
14286         {
14287             spCalendarFolder = m_spCDOSession->GetDefaultFolder((long)CdoDefaultFolderCalendar);
14288         }
14289
14290         // Find state info, if it exists, for this item using refid
14291         lAppRefId = 0;
14292         lAppParentRefId = 0;
14293         lAppDeviceSeqNum = 0;
14294         spAppointmentItem = 0;
14295         spAppointmentFields = 0;
14296         spAppointmentSourceKey = 0;
14297         if( pbInStateSourceKey )
14298         {
14299             MAPIFreeBuffer(pbInStateSourceKey);
14300             pbInStateSourceKey = 0;
14301         } // if
14302
14303         /* Get the UID (our ref Id) */
14304         RIMEvent:::UIDProperty* pAppointmentUID;
14305         if( !CalendarEvent.GetProperty( &pAppointmentUID ) )
14306         {
14307             DebugLog:::Printf( DebugLog:::LOG_WARNING,
14308                             "MAPIMailbox::Synchronize - Getting UIDProperty from RIMCalendar failed for %s.",
14309                             m_Name );
14310             bResult = false;
14311             goto __end;
14312         }
14313         lAppRefId = pAppointmentUID->GetUID();
14314
14315         /* Get the RelatedTo Id */
14316         RIMEvent:::RelatedToIDProperty* pAppointmentRelatedTo;
14317         if( CalendarEvent.GetProperty( &pAppointmentRelatedTo ) )
14318         {
14319             lAppParentRefId = pAppointmentRelatedTo->GetRelatedToID();
14320             RIMEvent:::RecurrenceIDProperty* pRecurrenceID;
14321             if( CalendarEvent.GetProperty( &pRecurrenceID ) )
14322             {
14323             }
14324         }
14325     }

```



```

14365     continue;
} // If not filtered, use source key (and possibly instance date) to retrieve item
        // using MAPI. Get the start and end time to generate a CDO filter.

14370     // Get contents table for calendar folder
hResult = m_pFolders[ CALENDAR ]->GetContentsTable( 0, &pContentsTable );
if( CallFailed( hResult ) )
{
    ReportError( DebugLog::LOG_WARNING, mailbox_fcn, "GetContentsTable", hResult );
    bResult = false;
    goto __end;
}

} // if

14375     // Set properties we are interested in.
hResult = pContentsTable->SetColumns( reinterpret_cast<SPropTagArray*>( &sptaAppointmentProps ), 0 );
if( CallFailed( hResult ) )
{
    ReportError( DebugLog::LOG_WARNING, mailbox_fcn, "SetColumns", hResult );
    bResult = false;
    goto __end;
}

14380     // Restrict the rows returned to only those with a matching Source Key value.
sSourceKeyRes.rt = RES_CONTENT;
sSourceKeyRes.res.resContent.ulFuzzyLevel = FL_FULLSTRING;
sSourceKeyRes.res.resContent.ulPropTag = PR_SOURCE_KEY;
spvSourceKey.ulPropTag = PR_SOURCE_KEY;
spvSourceKey.Value.bin.cb = pbinStateSourceKey->cb;
spvSourceKey.Res.resProperty.ipb = pbinStateSourceKey->ipb;
sSourceKeyRes.res.resProperty.ipProp = &spvSourceKey;
hResult = pContentsTable->Restrict( &sSourceKeyRes, TBL_BATCH );
if( CallFailed( hResult ) )
{
    // Query failed.
    ReportError( DebugLog::LOG_WARNING, mailbox_fcn, "Restrict", hResult );
    goto __end;
}

14385     // Get all matching rows. Should only be one, otherwise error.
hResult = HrQueryAllRows( pContentsTable, 0, 0, 0, &pRows );
}

```

14390

14395

14400

14405

```

    if( CallFailed( hResult ) )
    {
        ReportError( DebugLog::LOG_WARNING, mailbox_fcn, "HrQueryAllRows", hResult );
        goto __end;
    } // if

    if( pRows->cRows != 1 )
    {
        // error: should only be one row returned
    }

    // If this is a root appointment, indicated by no related-to id, use start and end time for filter
    if( !lAppParentRefId && !lStateParentRefId )
    {
        fStartTimeFilter = pRows->aRow[ 0 ].lpProps[ 4 ].Value.ft;
        fEndTimeFilter = pRows->aRow[ 0 ].lpProps[ 5 ].Value.ft;
    }
    // Else if this is exception
    else
    {
        // Open the root appointment
        pRootAppointment = OpenMessage( reinterpret_cast<LPENTRYID>(pRows->aRow[ 0 ].lpProps[ 0 ].Value.bin.lpB),
                                         pRows->aRow[ 0 ].lpProps[ 0 ].Value.bin.cb,
                                         CALENDAR );
    }

    // Get the attachment table for this appointment

    hResult = pRootAppointment->GetAttachmentTable( 0, &pAttachTable );
    if( CallFailed( hResult ) )
    {
        ReportError( DebugLog::LOG_WARNING, mailbox_fcn, "GetAttachmentTable", hResult );
        bResult = false;
        goto __end;
    } // if

    // Set properties we are interested in.
    hResult = pAttachTable->SetColumns( reinterpret_cast<SPropTagArray*>( &spatAttachProps ), 0 );
    if( CallFailed( hResult ) )
    {
        ReportError( DebugLog::LOG_WARNING, mailbox_fcn, "SetColumns", hResult );
        bResult = false;
    }

    14440
    14445

```

```

14450      goto __end;
14451      } // if

14452      // *****
14453      // *** Process Attachments ***
14454      // *****

14455      // Process all attachments.
14456      hResult = HrQueryAllRows( pAttachTable, 0, 0, 0, &pAttachRows );
14457      if( CallFailed( hResult ) )
14458      {
14459          ReportError( DebugLog::LOG_WARNING, mailbox_fn, "HrQueryAllRows", hResult );
14460          bResult = false;
14461          goto __end;
14462      } // if

14463      // Call GetInstanceFilter to get start & end time for CDO filter, instance is Recurrence-ID
14464      UnixTimeToFileTime(InstanceDate, flInstanceDate );
14465      GetInstanceFilter( pRootAppointment, pAttachRows, pAttachRows->cRows,
14466                         flInstanceDate, &flStartTimeFilter, &flEndTimeFilter );
14467      }

14468      // Using CDO filter, obtain appt object and update using values contained in the RIMCalendar
14469      GetCDOAppointment( spAppointment, spCalendarFolder,
14470                         flStartTimeFilter, flEndTimeFilter,
14471                         pRows->arRow[0].lpProps[2].Value.l );
14472      }

14473      else
14474      {
14475          // Else if no state record for this item, then it's a new record
14476          // If the item is a child appointment, indicated by Related-To property
14477          if( pAppointmentMethod->GetMethod() == RIMCalendar::CANCEL )
14478          {
14479              // No state info, so can't delete it. Skip and go to next event.
14480          }

14481          if( lApptParentRefId )
14482          {
14483              // Retrieve state info for parent. Will need to identify instance and to update state.
14484              // Search for state info corresponding to the parent refId
14485              hResult = MAPIAlocateBuffer( sizeof( SBinary ), reinterpret_cast<VOID*>( &pbmStateSourceKey ) );
14486          }

```

```

if( CallFailed( hResult ) )
{
    ReportError( DebugLog::LOG_WARNING, mailbox_fcn, "MAPIMailbox::AllocateBuffer", hResult );
    bResult = false;
    goto __;
} // if

m_CalSyncStateMutex.Lock();
bFound = m_pCalendarState->QueryByRefId( lAppParentRefId, pbInStateSourceKey,
                                            &IsStateParentRefId, &IsState_LastModified,
                                            &IsStateDeviceSeqNum, &IsStateDesktopSeqNum,
                                            &fStateDellInProgress, &fStateModInProgress );
m_CalSyncStateMutex.Unlock();
}

14495 // Search for parent item. Table columns will be source key, inet article #, duration, last mod
// Restrict the rows returned to only those with a matching Source Key value.
sSourceKeyRes.rt = RES_CONTENT;
sSourceKeyRes.resContent.ulFuzzyLevel = FL_FULLSTRING;
sSourceKeyRes.resContent.ulPropTag = PR_SOURCE_KEY;
spvSourceKey.ulPropTag = PR_SOURCE_KEY;
spvSourceKey.Value.bin.cb = pbInStateSourceKey->cb;
spvSourceKey.Value.bin.lpbf = pbInStateSourceKey->lpbf;
sSourceKeyRes.resPropertyParams.lpProp = &spvSourceKey;
hResult = pContentsTable->Restrict(&sSourceKeyRes, TBL_BATCH);
if( CallFailed( hResult ) )
{
    // Query failed.
    ReportError( DebugLog::LOG_WARNING, mailbox_fcn, "Restrict", hResult );
    goto __;
} // if

// Get all matching rows. Should only be one, otherwise error.
hResult = HrQueryAllRows( pContentsTable, 0, 0, 0, 0, &pRows );
if( CallFailed( hResult ) )
{
    ReportError( DebugLog::LOG_WARNING, mailbox_fcn, "HrQueryAllRows", hResult );
    goto __;
} // if

if( pRows->cRows != 1 )
{
    // error: should only be one row returned
}

14500
14505
14510
14515
14520
14525
14530

```

```

14535     }
    // Build CDO filter using date/time in Recurrence-ID property, plus duration
    UnixTimeToFileTime(lInstanceDate, ftStartTimeFilter);
    UnixTimeToFileTime((lInstanceDate+pRows->aRow[0].lpProps[2].Value.l*60), ftStartTimeFilter);
    // Retrieve appi object
    GetCDOAppointment( spAppointmentItem, spCalendarFolder,
                        ftStartTimeFilter, ftEndTimeFilter,
                        pRows->aRow[0].lpProps[2].Value.l );
14540
14545     // Else
     {
        // New stand alone appointment. Create appi object.
        spCalendarMessages = spCalendarFolder->GetMessages();
        spAppointmentItem = spCalendarMessages->Add();
     }
14550
14555     // If this is a PUBLISH method
     if( pAppointmentMethod->GetMethod() == RIMCalendar::PUBLISH )
     {
        RIMEventtoCDOAppointment(&CalendarEvent, spAppointmentItem);
        spAppointmentItem->Update(true,true);
     }
14560
14565     // Update state info
     spAppointmentSourceKey = spAppointmentFields->GetItem((long)PR_SOURCE_KEY);
     vtAppointmentSourceKey = spAppointmentSourceKey->GetValue();
     vtLastModified = spAppointmentItem->GetTimeLastModified();
     if((VariantTimeToSystemTime(vtLastModified, &stLastModified) ||
        !SystemTimeToFileTime(&stLastModified, &vtLastModified)))
     {
        DebugLog::Printf( DebugLog::LOG_WARNING,
                         "MAPIMailbox::RIMMeetingToMAPIMeeting - Converting VariantTime to FileTime Failed." );
        bResult = false;
        goto __end;
     }
14570
14575     if( !(pbinStateSourceKey->lpb) )
     {
        // JAG Note: Redo this whole thing! Uckkk!
        hResult = MAPIMailbox::RIMMeetingToMAPIMeeting( strin_bstr_t(vtAppointmentSourceKey))/2,
                  pbinStateSourceKey,
                  reinterpret_cast<VOID*> ( &(pbinStateSourceKey->lpb) ));
        if( CallFailed( hResult ) )

```

```

14580 {
14581     ReportError( DebugLog::LOG_WARNING, mailbox_fcn, "MAPIMailbox::AllocateMore", hResult );
14582     bResult = false;
14583     goto __end;
14584 } //if
14585 if(!FBinFromHex(_bstr_t(vtAppointmentSourceKey), pbInStateSourceKey->pb))
14586 {
14587     //error
14588 }
14589 pbInStateSourceKey->cb = strlen(_bstr_t(vtAppointmentSourceKey))/2;
14590 m_CalSyncStateMutex.Lock();
14591 m_pCalendarState->UpdateStateRecord(lAppRefId, pbInStateSourceKey, 0,
14592                                         lAppParentRefId ? &flInstanceDate : 0,
14593                                         lAppParentRefId,
14594                                         &flLastModified,
14595                                         lAppDeviceSeqNum, 0, false, false );
14596 }
14597 // Else if this is a CANCEL method
14598 else if( pAppointmentMethod->GetMethod() == RIMCalendar::CANCEL )
14599 {
14600     spAppointmentItem->Delete();
14601     spAppointmentItem = 0;
14602     // Delete any state info for this item
14603     m_CalSyncStateMutex.Lock();
14604     m_pCalendarState->DeleteByRefId( lAppRefId );
14605     m_CalSyncStateMutex.Unlock();
14606 }
14607 pCalendarEvent = 0;
14608 pAppointment->GetNextProperty( &pCalendarEvent );
14609 } // while
14610 }
14611 catch( ... )
14612 {
14613     ReportException(mailbox_fcn);
14614     bResult = false;
14615     goto __end;
14616 }
14617

```

```
    } // Exception handling
}

14625    try
14626    {
14627        // Cleanup.
14628        if( pbInStateSourceKey )
14629        {
14630            MAPIFreeBuffer( pbInStateSourceKey );
14631        } // if
14632
14633        return( bResult );
14634    }
14635    catch( ... )
14636    {
14637        // Cleanup failed; not critical (we hope).
14638        ReportException( mailbox_fcn );
14639
14640        return( bResult );
14641
14642        } // if
14643
14644    #else
14645        // Desktop redirector somehow called calendar method. Error.
14646        DebugLog::Printff( DebugLog::LOG_ERROR,
14647            "MAPIMailbox::Synchronize - Desktop Redirector accessed calendar method." );
14648
14649        return( false );
14650    #endif
14651
14652
14653    }
14654
14655    MAPIMailbox.cpp
```



Invention Disclosure Form

Page 1 of 9
DISCLOSURE DATE

Reference Number:
TO BE ASSIGNED BY:

03/13/00 4:53 PM 02/21/00 6:09

THE PURPOSE OF THIS FORM IS TO:

- bring to Chuck Meyer's attention the development of ideas that are believed to be novel and of value to RIM;
 - provide a description of the Invention that establishes an invention date;
 - provide RIM's outside patent counsel with a description of the Invention that facilitates processing of a patent application.

Complete this form by typing the information requested. Submit the completed disclosure form with any additional material requested below to Chuck Meyer.

1. **TITLE OF INVENTION:** Enter a short descriptive phrase that will serve as the title of the Invention.

THE ST. LUCAS CHURCH OF THE HOLY SPIRIT

2. DESCRIPTION OF INVENTION: Enter a brief description of the nature and application of the Invention.

This invention will allow "over the air" changes on a PDA calendar to be synchronized with the data on the host calendar. This includes all types of calendar entries, including recurring appointments and exceptions to a recurrence. All operations will be supported: creation, modification, and deletion of appointment entries. Conflicts that occur during synchronization are resolved without user interaction based on a simple [REDACTED] model where an end point is designated the authority in any dispute. [REDACTED] calendar component will be used to implement the synchronization of the calendar synchronization feature. The process will allow for [REDACTED] calendar. There will be a [REDACTED] option to stop the synchronization of calendar events.

3. Identify the projects and products where the Invention may be incorporated.

Blackberry

Proton

Future PDA products

4. The following dates apply to the Invention:

Inventors:

Hugh Hind

FULL NAME (TYPE)

Craig Dunk

This Invention has been witnessed
and understood by me:

FULL NAME (TYPE)

SIGNATURE

Digitized by srujanika@gmail.com



Invention Disclosure Form

Page 2 of 9

Reference Number:
TO BE ASSIGNED BY:
03/13/00 4:53 PM 02/21/00 6:00

5. Describe the concern or problem your Invention addresses.

Currently the device provides a calendar application where users may maintain appointment and event-related information. Information stored on the host and device are distinct, with the synchronization of the two separate databases only occurring as part of a serial synchronization process when the pager is placed in the docking cradle. When the device is connected to the host, the host will automatically synchronize the two databases. This process must be able to synchronize both databases in one step. Conflicts can be avoided with the use of a merge routine; however, this process will use only one step to reduce the number of data transfers between the device and host.

6. Identify any devices or systems known to you that address the same concern or problem. (Also include those developed by RIM and any known patents or publications that address the problem (attach copies).

7. Identify the differences between your Invention and each known device or article described above and describe any impact those differences can have in providing value to

N/A

8. Identify any other unique aspects of your Invention.

9. Provide a detailed description of your Invention. Attach any drawings, circuit diagrams, or other documentation that describes the Invention; where appropriate, apply reference numerals to drawings and use those reference numerals in your detailed description. You must disclose the best mode you know for practicing the Invention.

Inventors:

Hugh Hind

FULL NAME (TYPE)

SIGNATURE

Date

Craig Dunk

FULL NAME (TYPE)

SIGNATURE

Date

This Invention has been witnessed
and understood by me:

FULL NAME (TYPE)

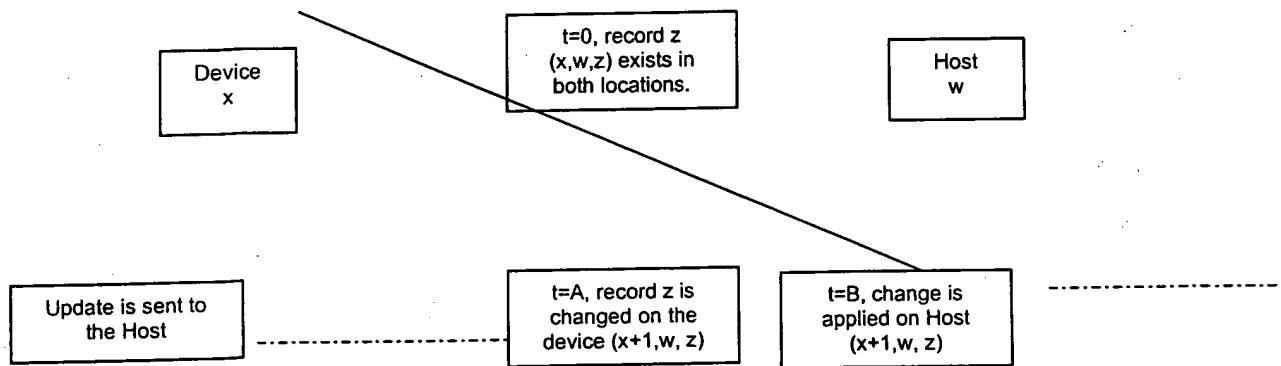
SIGNATURE

Date

Page 3 of 9
DISCLOSURE DATE:

Reference Number:
TO BE ASSIGNED BY:
03/13/00 4:53 PM 02/21/00 6:09

Fig 1: Calendar Record Synchronization for a Device and one Host



Firstly let us consider the case when no conflict occurs. Each record entry has two version numbers: "host version" w and "device version" x . Both the device and host recognise these numbers in the record entry. At $t=0$ (where t is time), the device and host are synchronized and both contain the record which has the version numbers w, x and the record z number (e.g. (x, w, z)). Suppose the device makes a change. To illustrate, in Fig.1, at $t=A$, the device makes a change to the record z and updates the device version to $x+1$ associated with the record. Preferably, the updated record is sent via the wireless network to the host in order to synchronize the device and host. At $t=B$, the host makes the changes and updates its record z to $x+1$.

Conversely, when the host makes a change to record z , it updates the host version number to $w+1$ and sends the update via the wireless network to the device at $t=A$. At $t=B$, the device accepts the change and updates its host version number to $w+1$.

A conflict occurs when either the host or device makes a change to a record before $t=B$ (i.e., before notification of the change arrives to the device or host respectively after the change made at $t=A$). The version numbers are now out of sync. In this case one of the changes will be discarded based on the "master" setting which the user selected. This setting selects either the host or the device so that when a conflict arises the "master" will override all changes. No

Inventors:

Hugh Hind

FULL NAME (TYPE) SIGNATURE Date

Craig Dunk

FULL NAME (TYPE) SIGNATURE Date

This
and

An update conflict occurs when:
A change occurs on the Host ($w+1$) before $t=B$ and arrives to the device after $t=A$ (and vice versa)

No update conflict occurs when:
A change on the Host and an update on the device occur either entirely before $t=A$ or entirely after $t=B$

FULL NAME (TYPE) SIGNATURE Date

Page 4 of 9
DISCLOSURE DATE:

Reference Number:
TO BE ASSIGNED BY:

03/13/00 4:53 PM 02/21/00 6:09

conflicts occur if the "second" changes occurs on both the device or and host take place entirely before an update occurs, $t < A$ or entirely after the update occurs $t > B$.

Fig 2.1: Conflict when Host is Master
 $t=A$



$t=B$



For example, in Fig 2.1, the host is set as the master. At $t=A$ (where t is time), the host makes a change to record z , increments the host version number to $w+1$, and sends an update to the device. However, the device has also made a change to record z , incremented the device version number to $x+1$, and sent an update to the host before receiving the host changes. The two records on the devices are now out of sync. At $t=B$, the host, being the master will discard the device changes. The device upon receipt of the host's update will accept the changes from the host and discard the changes the device made. The device will increment the host version number to $w+1$ and decrement the device version number back down to x .

Fig 2.2: Conflict Resolution when Device is Master



Conversely, in Fig 2.2, the device is set as the master. At $t=A$, both the device and host make an update at the same time, incrementing their respective version numbers to $x+1$ and $w+1$. At $t=B$, the device, being the master, will discard the host's changes. The host, upon receipt of the device's update, will discard the previous change the host made, and accept the device's changes. The host will increment the device version number to $x+1$ and decrement the host version number to w .

Inventors:

Hugh Hind

FULL NAME (TYPE)

SIGNATURE

Date

Craig Dunk

FULL NAME (TYPE)

SIGNATURE

Date

This Invention has been witnessed
and understood by me:

FULL NAME (TYPE)

SIGNATURE

Date

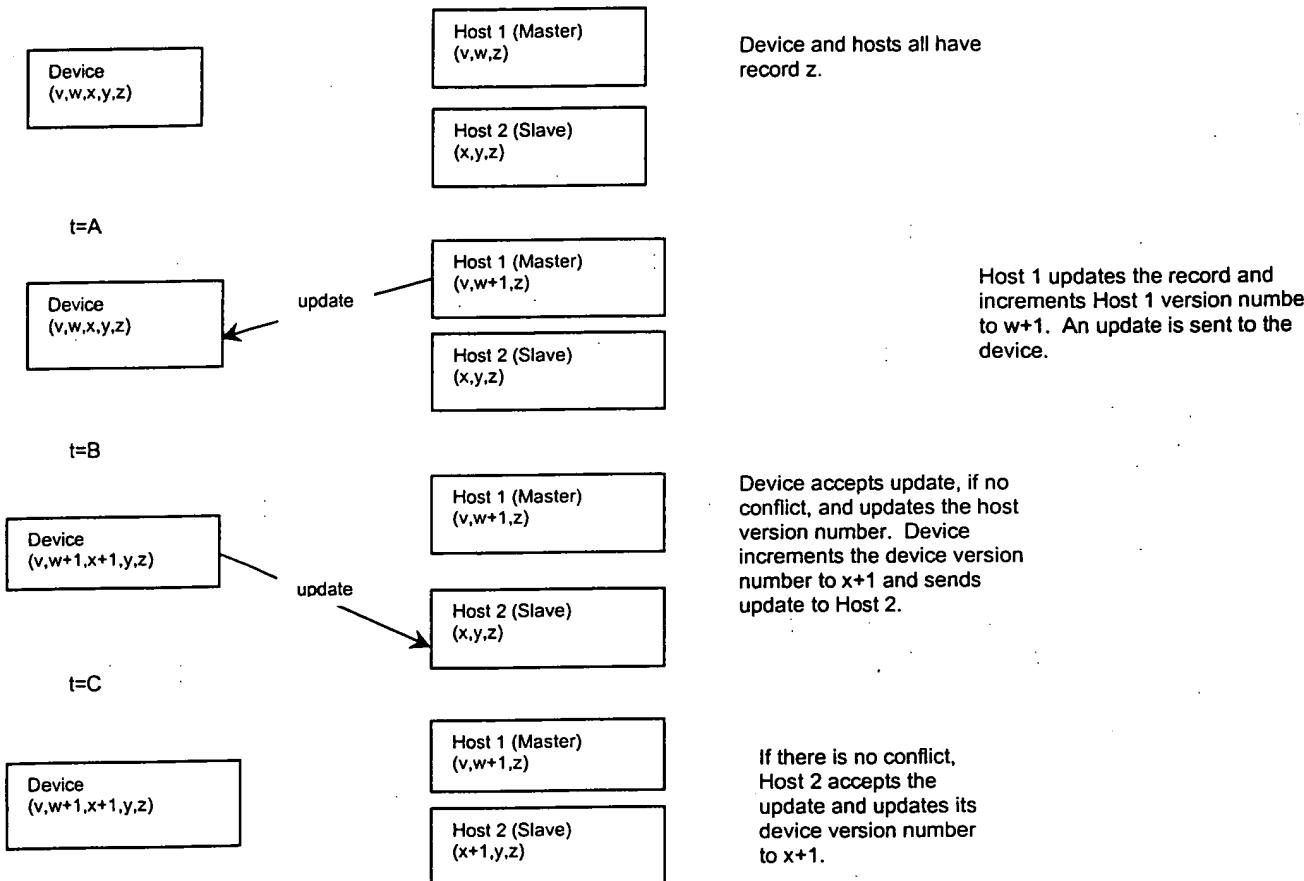


Invention Disclosure Form

Page 5 of 9
DISCLOSURE DATE:

Reference Number:
TO BE ASSIGNED BY:
03/13/00 4:53 PM 02/21/00 6:09

Fig 3: Multiple Hosts – Update sent from Master Host
t=0



The device may also communicate with more than one host (as in Fig. 3). In the preferred embodiment, the hosts do not communicate with each other. In this embodiment, one of the hosts is set as the "master". The device recognizes this setting. The other hosts, or "slave" hosts, recognize the device as their "master".

Indices are scrambled in this paragraph ... In Fig. 3, at t=0 (where t is time), the device and hosts all contain the record. The record is represented by a number z. All hosts have a unique host version number and a unique device version number. The diagram illustrates an example of one device and two-host system. The master host contains the number (v,w,z). The number "w" is the host version number. The master host also has a device version number "v".

Inventors:

Hugh Hind

FULL NAME (TYPE)

SIGNATURE

Date

Craig Dunk

FULL NAME (TYPE)

SIGNATURE

Date

This Invention has been witnessed
and understood by me:

FULL NAME (TYPE)

SIGNATURE

Date



Invention Disclosure Form

Page 6 of 9
DISCLOSURE DATE:

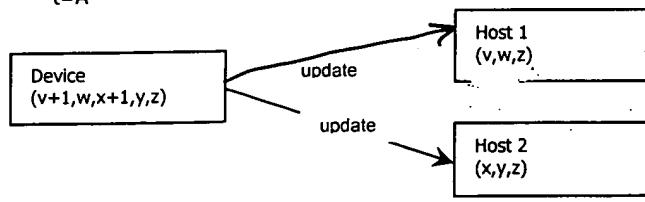
Reference Number:
TO BE ASSIGNED BY:

03/13/00 4:53 PM 02/21/00 6:09

The slave host contains the number (x,y,z). The slave host has a host version number "y" and its own device version number "x". The device contains all these numbers (v,w,x,y,z)

At t=A, the master host updates the record z. The master host then increments its version number to w+1 and sends an update to the device via the wireless network. At t=B, if there is no conflict, the device accepts the change and increments the host version number to w+1. The device also increments the device version number that is associated with the slave host to x+1 and sends an update to the slave host. At t=C, the slave host accepts the change and increments its device version number to x+1, as long as there is no conflict.

Fig 4: Multiple Hosts--Update from the Device
t=A



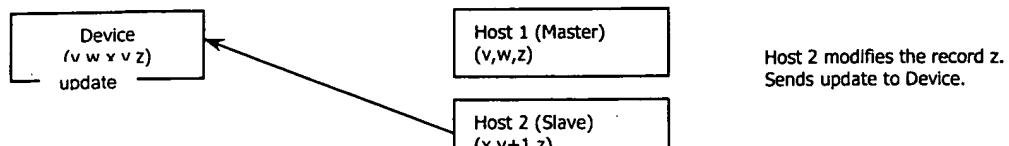
t=B



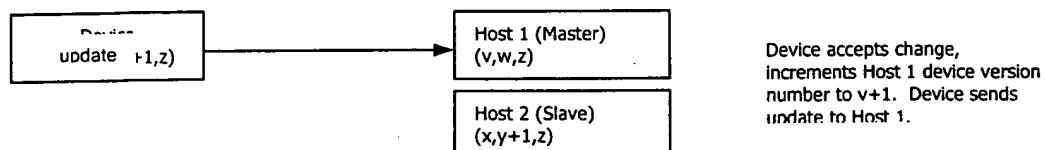
Fig 4 illustrates when the device updates the record z. At t=A, it will update the device version numbers to v+1 and x+1. The device sends an update to both hosts over the wireless network. At t=B, if there is no conflict with the hosts, they accept it and update their own device version numbers accordingly. In this situation, the host or the device could be the master.

Fig 5: Multiple Hosts--Update from the Slave Host

t=A



t=B



t=C

Inventors:

Device
(v+1,w,x,y+1,z)

FULL NAME (TYPE)

Craig Dunk

FULL NAME (TYPE)

Host 1 (Master)
(v+1,w,z)

RE

Date

Host 2 (Slave)
(x,y+1,z)

SIGNATURE

Date

This Invention has been witnessed
and understood by me:

FULL NAME (TYPE)

SIGNATURE

Date

Page 7 of 9
DISCLOSURE DATE:

Reference Number:
TO BE ASSIGNED BY:
03/13/00 4:53 PM 02/21/00 6:09

Host 1 accepts change

Fig 5 illustrates the situation where the slave host makes an update. At $t=A$ (where t is time), the slave host has changed the record z , increments its host version number y to $y+1$, and sends an update over the wireless network to the device. At $t=B$, the device accepts the modifications and changes the record. The device increments the master host device number from v to $v+1$, and sends an update to the master host. At $t=C$, the master host accepts the change and increments its device version number to $v+1$.

Fig 6: Multiple Hosts -- Conflict Resolution Between Master and Device

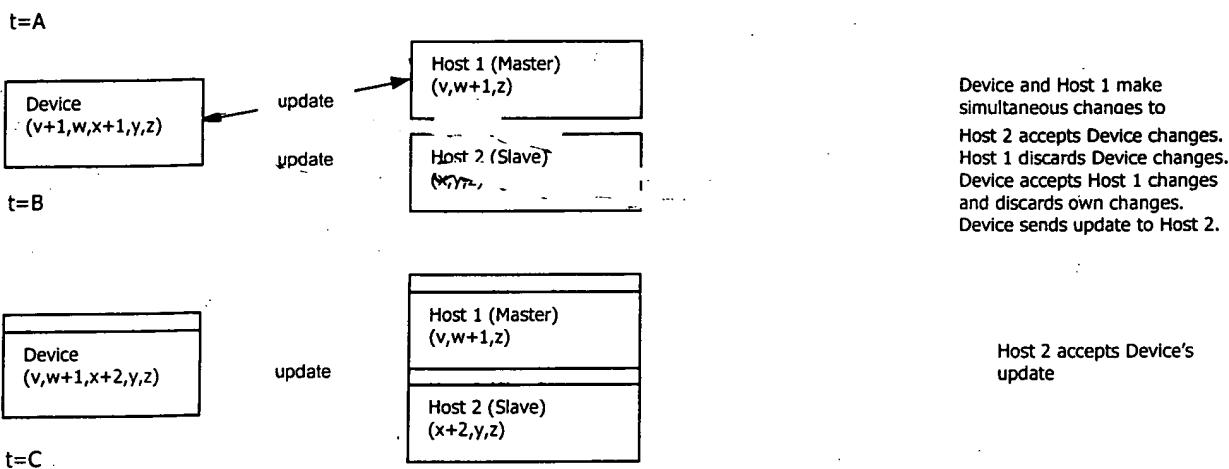
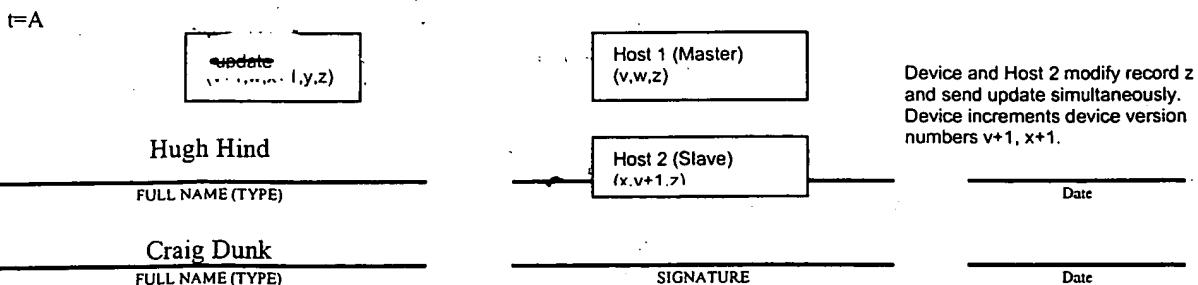


Fig 6 illustrates a conflict resolution in a situation with multiple hosts. At $t=A$ (where t is time), the master host and device make changes to record z . The device increments the device version numbers to $v+1$ (for the master host) and to $x+1$ (for the slave host). The master host increments the master host version number from w to $w+1$. The master host sends an update over the wireless network to the device. The device simultaneously sends an update to the master host and slave host. At $t=B$, the slave host accepts the device changes and increments its device version number to $x+1$. The master host discards the device changes. Upon receipt of the master host changes, the device discards its earlier changes and accepts the master host changes. The device decrements the master host device version number back to v , increments the master host host version number to $w+1$, and increments the slave host device version number to $x+2$. The device sends an update to the slave host. At $t=C$, the slave host accepts the device changes and increments its device version number to $x+2$.

Fig 7: Multiple Hosts --Conflict Resolution between Slave Host and Device



Inventors:

Hugh Hind

FULL NAME (TYPE)

Host 1 (Master)
(v, w, z)

Date

Craig Dunk

FULL NAME (TYPE)

Host 2 (Slave)
(x, v+1, z)

SIGNATURE

Date

This Invention has been witnessed
and understood by me:

FULL NAME (TYPE)

SIGNATURE

Date



Invention Disclosure Form

Page 8 of 9
DISCLOSURE DATE:

Reference Number:
TO BE ASSIGNED BY:
03/13/00 4:53 PM 02/21/00 6:09

update

t=B

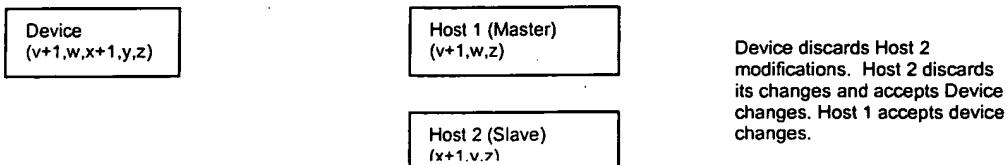
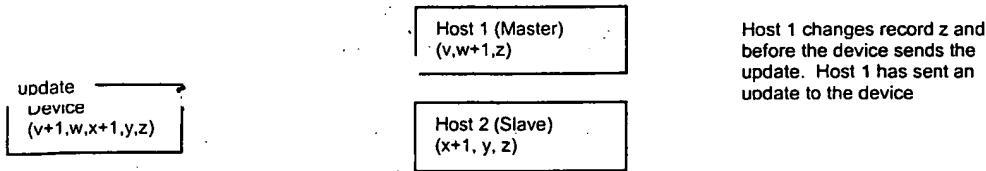


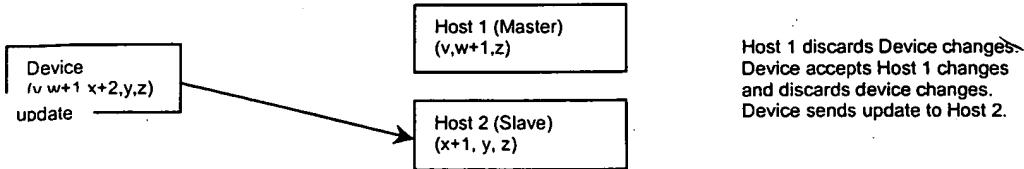
Fig 7.2 illustrates a conflict resolution between the device and slave host when the slave host sends out an update. At t=A, the slave host changes the record z and increments its host version number to y+1. The device also changes the record and increments the device numbers for the master and slave hosts to v+1 and x+1 respectively. Both the slave host and device send and update via the wireless network simultaneously. There now exists a conflict between the record that the device holds and the update that was sent to the device. Because the device acts as a "master" to the slave host, at t=B, the device discards the changes that the slave host made. An update is sent from the device to the slave host and the slave host discards the previous update and accepts the device's changes. The device sends an update to the master host. The master host will make the necessary updates as long as there is no conflict. The slave host decrements the host version number back to y and increments the device version number to x+1. The device then sends an update to the master host. The master host will make the necessary updates as long as there is no conflict.

Fig 7.2: Multiple Hosts --Conflict between Device and Master Host after Slave Host has sent an update.

t=B



t=C



t=D

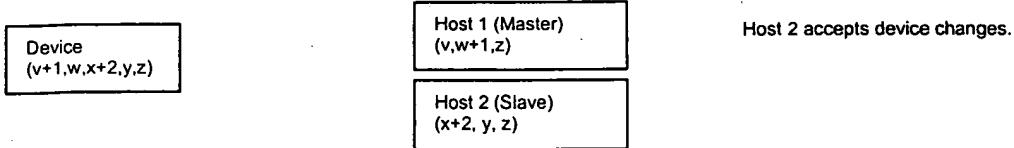


Fig 7.2 continues from Fig 7.1. At t=B, if the master host changes the record (and increments its host version number to w+1) before it receives the device's update, there is now a conflict. At t=C, the master host will discard the update from the device. The device also discards the changes that it made and accepts the master host's update. The device decrements the master host device

Inventors:

Hugh Hind

FULL NAME (TYPE)

SIGNATURE

Date

Craig Dunk

FULL NAME (TYPE)

SIGNATURE

Date

This Invention has been witnessed
and understood by me:

FULL NAME (TYPE)

SIGNATURE

Date

Page 9 of 9
DISCLOSURE DATE:

Reference Number:
TO BE ASSIGNED BY:

03/13/00 4:53 PM 02/21/00 6:09

version number back to v , and increments the slave host device version number to $x+2$. The device sends an update to the slave host. The slave host accepts the update and increments its device version number.

Fig. 8: Multiple Hosts – Conflict Resolution when the Device is Master
 $t=A$

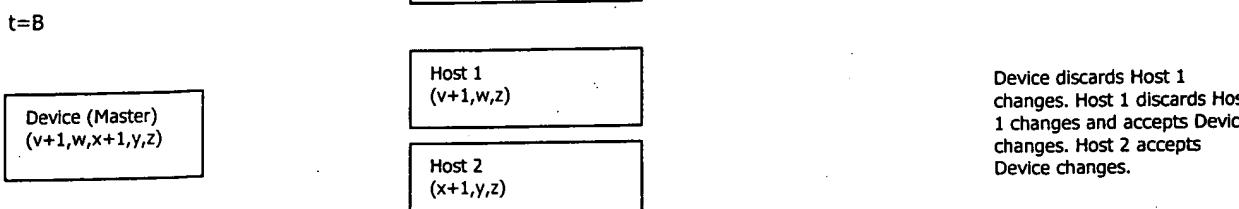
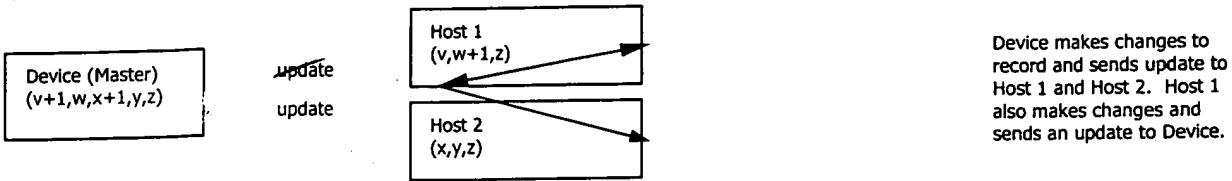


Fig 8 illustrates the situation where the device has the "master" setting for all the hosts and has a conflict with either host. At $t=A$, the device makes a change to record z . The device increments the device version numbers for Host 1 and Host 2 to $v+1$ and $x+1$ respectively. Then the device sends an update to the hosts over the wireless network. Host 1 makes changes to record z and increments its host version number to $w+1$. Host 1 then sends out an update at the same time as the device or before Host 1 has received the device's update. The host 1 record and the device record are out of sync. At $t=B$, host 2 accepts the Device changes and increments its device version number to $x+1$. The device, because it is set as the master, discards Host 1's changes. Host 1 accepts the device changes and discards its own changes. Host 1 increments the device version number to $v+1$ and decrements its host version number to w .

Inventors:

Hugh Hind

FULL NAME (TYPE)

SIGNATURE

Date

Craig Dunk

FULL NAME (TYPE)

SIGNATURE

Date

This Invention has been witnessed
and understood by me:

FULL NAME (TYPE)

SIGNATURE

Date